

PROGRESS

Introducción

Base de Datos

Base de Datos: es una colección de información interrelacionada así como un conjunto de programas para tener acceso a dicho datos, usada para las aplicaciones de una organización. El objetivo de una base de datos es proporcionar un entorno que sea conveniente y eficiente para extraer y almacenar información.

Algunos de los problemas que se presentan cuando no se utiliza una base de datos son:

- **Redundancia e inconsistencia de los datos:** Debido a que los archivos de programas de aplicaciones son creados por diferentes personas a través del tiempo, puede ocurrir que los archivos tengan diferente formato o se encuentren duplicados, de la misma manera puede ocurrir que los programas para acceder a la información se encuentre en más de un sitio.
- **Desarrollos de múltiples aplicaciones para la recuperación de la información:** Un sistema de información basado en Base de datos pretende desarrollar aplicaciones de uso general para la recuperación de la información, tratando de establecer estándares tanto en las aplicaciones como en la forma de almacenar los datos.
- **Proliferación de archivos:** Pretende aislar los datos de las aplicaciones para un mejor control de tal manera que no se tenga que modificar la estructura de la Base de datos cuando se desarrolle una nueva aplicación.
- **Evitar el acceso simultaneo de los usuarios a la misma información:** Control de acceso a la información ara evitar su inconsistencia, permitiendo que solamente un usuario tenga acceso a los datos y colocando candados hasta que estos sean liberados.
- **Problemas de seguridad:** Implantar las restricciones de seguridad para tener acceso a los datos.
- **Problemas de integridad:** Colocar restricciones de consistencia de datos.

Sistema de información: Conjunto de elementos interrelacionados que tiene como fin suministrar a los usuarios la información requerida para la toma de decisiones.

Sistema administrador de Base de datos(Data Base Manager System): Es un conjunto de herramientas integrado que proporciona el entorno para un eficiente almacenamiento y recuperación de la información.

Abstracción de los datos

Un objetivo importante en un sistema de Base de datos es proporcionar a los usuarios una visión abstracta de los datos; el sistema esconde ciertos detalles de cómo se almacenan y mantiene los datos.

- **Nivel físico:** En este nivel se describe como están los datos realmente almacenados, se describen en detalle las estructuras de datos complejas.
- **Nivel conceptual:** En este nivel se describe que datos son almacenados realmente y las relaciones que existen entre ellos. Este nivel lo emplean los administradores de la Base de datos ya que ellos deben decidir que información es la que se almacena.
- **Nivel de visión:** Este nivel es empleado por los usuarios ya que ellos solamente necesitan ver los datos con los que va a trabajar.

MODELO DE DATOS

El modelo de datos es un conjunto de herramientas conceptuales para describir datos, las relaciones entre ellos, la semántica asociada entre ellos y las restricciones de consistencia.

Modelos lógicos basado en objetos: Se usan para describir a los datos en los niveles de conceptual y de visión entre los mas conocidos se encuentran:

- Modelo Entidad-Relación.
- Modelo Orientado a Objetos.
- Modelo Binario.
- Modelo Semántico de Datos.
- Modelo Infológico.
- Modelo Funcional de datos.

Modelo Entidad-Relación

Se basa en una percepción del mundo real que consiste en un conjunto de objetos básicos llamados entidades y las relaciones entre ellos.

Entidades

Son objetos fundamentales del mundo real acerca de los cuales se almacena información. Estos objetos pueden ser distinguibles mediante sus características llamadas atributos que los hacen únicos. Se clasifican en dos grupos:

1. *Entidades fuertes*: Tienen identificadores que determinan de una manera única a las ocurrencias de las entidades. Por lo tanto se puede afirmar que tienen existencia de otras entidades.
2. *Entidades débiles*: Su existencia depende de los atributos de una o más entidades “padres” por lo tanto su existencia está limitada por la existencia de las entidades padre.

Atributos: Son las características que poseen las entidades mediante las cuales se pueden describir y que tomadas todas en conjunto hacen que dicha entidad sea única.

Relación: Es la asociación entre las entidades. La relación puede ser Simple o Compleja.

- Simple: Es cuando existe una relación de uno a uno.
- Compleja: Es cuando existe una relación de 1:M, M:1, M:M.

Diagrama Entidad-Relación

Es la representación gráfica lógica global de una base de datos, puede representarse gráficamente por medio de un diagrama de E-R que consta de los siguientes componentes:

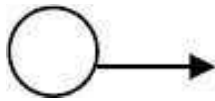
Entidades :



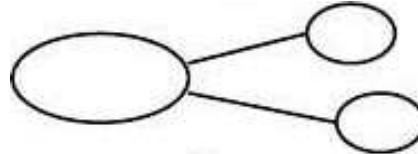
Entidad Fuerte



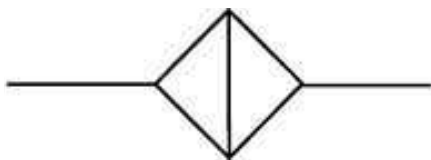
Entidad Débil



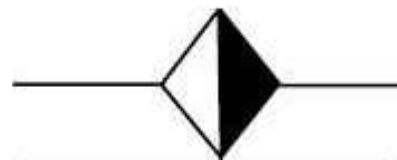
Atributo



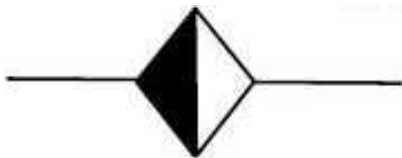
Atributo Compuesto



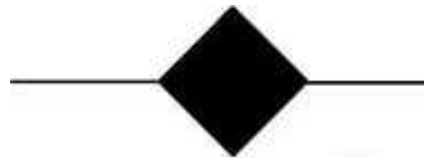
Relación de 1:1



Relación de 1:M



Relación de M:1



Relación M:M

Reglas de interrelación de tablas:

Relación 1:1

La llave primaria de una de las dos tablas o entidad pasa a la siguiente tabla o entidad.

Relación 1:M, M:1

La llave primaria de la tabla que tiene la relación número 1 pasa como llave foránea a la tabla de relación de muchos.

Relación M:M

Se formara una nueva tabla que contenga las llaves primarias de ambas tablas o entidades y estas formaran una llave primaria en la nueva tabla, separadamente serán llaves foráneas.

Proceso de normalización

- El proceso de normalización es un estándar que consiste, básicamente, en un proceso de conversión de las relaciones entre las entidades.

Tomando como referencia la tabla siguiente:

AUTORES Y LIBROS				
NOMBRE	NACION	CODLIBRO	TITULO	EDITOR
Date	USA	999	IBD	AW
Ad.Mig.	ESP	888	CyD	RM
Ma.Piat.	ITA	777	CyD	RM
Date	USA	666	BdD	AW

Se plantean una serie de problemas:

- Redundancia: cuando un autor tiene varios libros, se repite la nacionalidad.
- Anomalías de modificación: Si Ad.Mig. y Ma.Piat. desean cambiar de editor, se modifica en los 2 lugares. A priori no podemos saber cuántos autores tiene un libro. Los errores son frecuentes al olvidar la modificación de un autor. Se pretende modificar en un sólo sitio.
- Anomalías de inserción: Se desea dar de alta un autor sin libros, en un principio. NOMBRE y CODLIBRO son campos clave, una clave no puede tomar valores nulos.

Asegurando:

- Integridad entre los datos: consistencia de la información.

El proceso de normalización nos conduce hasta el modelo físico de datos y consta de varias fases denominadas formas normales, estas formas se detallan a continuación.

Definición de la clave

Antes de proceder a la normalización de la tabla lo primero que debemos de definir es una clave, esta clave deberá contener un valor único para cada registro (no podrán existir dos valores iguales en toda la tabla) y podrá estar formado por un único campo o por un grupo de campos.

En la tabla de alumnos de un centro de estudios no podemos definir como campo clave el nombre del alumno ya que pueden existir varios alumnos con el mismo nombre. Podríamos considerar la posibilidad de definir como clave los campos nombre y apellidos, pero

estamos en la misma situación: podría darse el caso de alumnos que tuvieran los mismo apellidos y el mismo nombre (Juan Fernández Martín).

La solución en este caso es asignar un código de alumno a cada uno, un número que identifique al alumno y que estemos seguros que es único.

Una vez definida la clave podremos pasar a estudiar la primera forma normal.

■ Primera forma normal (1NF)

Se dice que una tabla se encuentra en primera forma normal (1NF) si y solo si cada uno de los campos contiene un único valor para un registro determinado. Supongamos que deseamos realizar una tabla para guardar los cursos que están realizando los alumnos de un determinado centro de estudios, podríamos considerar el siguiente diseño:

Código	Nombre	Cursos
1	Marcos	Inglés
2	Lucas	Contabilidad, Informática
3	Marta	Inglés, Contabilidad

Podemos observar que el registro de código 1 si cumple la primera forma normal, cada campo del registro contiene un único dato, pero no ocurre así con los registros 2 y 3 ya que en el campo cursos contiene más de un dato cada uno. La solución en este caso es crear dos tablas del siguiente modo:

Tabla A	
Código	Nombre
1	Marcos
2	Lucas
3	Marta

Tabla B	
Código	Curso
1	Inglés
2	Contabilidad
2	Informática
3	Inglés
3	Informática

Como se puede comprobar ahora todos los registros de ambas tablas contienen valores únicos en sus campos, por lo tanto ambas tablas cumplen la primera forma normal.

Una vez normalizada la tabla en 1NF, podemos pasar a la segunda forma normal.

Segunda forma normal (2NF)

La segunda forma normal compara todos y cada uno de los campos de la tabla con la clave definida. Si todos los campos dependen directamente de la clave se dice que la tabla está es segunda forma normal (2NF).

Supongamos que construimos una tabla con los años que cada empleado ha estado trabajando en cada departamento de una empresa:

Código Empleado	Código Dpto.	Nombre	Departamento	Años
1	6	Juan	Contabilidad	6
2	3	Pedro	Sistemas	3
3	2	Sonia	I+D	1
4	3	Verónica	Sistemas	10
2	6	Pedro	Contabilidad	5

Tomando como punto de partida que la clave de esta tabla está formada por los campos código de empleado y código de departamento, podemos decir que la tabla se encuentra en primera forma normal, por tanto vamos a estudiar la segunda:

- 1. El campo nombre no depende funcionalmente de toda la clave, sólo depende del código del empleado.
- 2. El campo departamento no depende funcionalmente de toda la clave, sólo del código del departamento.
- 3. El campo años si que depende funcionalmente de la clave ya que depende del código del empleado y del código del departamento (representa el número de años que cada empleado ha trabajado en cada departamento)

Por tanto, al no depender todos los campos de la totalidad de la clave la tabla no está en segunda forma normal, la solución es la siguiente:

Tabla A	
Código Empleado	Nombre
1	Juan
2	Pedro
3	Sonia
4	Verónica

Tabla B	
Código Departamento	Dpto.
2	I+D
3	Sistemas
6	Contabilidad

Tabla C		
Código Empleado	Código Departamento	Años
1	6	6
2	3	3
3	2	1
4	3	10
2	6	5

Podemos observar que ahora si se encuentran las tres tabla en segunda forma normal, considerando que la tabla A tiene como índice el campo Código Empleado, la tabla B Código Departamento y la tabla C una clave compuesta por los campos Código Empleado y Código Departamento.

■ Tercera forma normal (3NF)

Se dice que una tabla está en tercera forma normal si y solo si los campos de la tabla dependen únicamente de la clave, dicho en otras palabras los campos de las tablas no dependen unos de otros. Tomando como referencia el ejemplo anterior, supongamos que cada alumno sólo puede realizar un único curso a la vez y que deseamos guardar en que aula se imparte el curso. A voz de pronto podemos plantear la siguiente estructura:

Código	Nombre	Curso	Aula
1	Marcos	Informática	Aula A
2	Lucas	Inglés	Aula B
3	Marta	Contabilidad	Aula C

Estudiemos la dependencia de cada campo con respecto a la clave código:

- Nombre depende directamente del código del alumno.
- Curso depende de igual modo del código del alumno.
- El aula, aunque en parte también depende del alumno, está mas ligado al curso que el alumno está realizando.

- ✓ Table Name
- ✓ Etiqueta
- ✓ Descripción
- ✓ Nombre para “DUMP”.
- ✓ TRIGGERS y Atributos de traducción.
- ✓ Validación de borrado.
- ✓ Hidden
- ✓ Frozen
- ✓ String Attrs.
- ✓ Data server
- ✓ Check CRC
- ✓ Overdrable

Propiedades de los archivos:

- **Hidden**: Oculta un archivo para evitar que sea desplegado en la lista de archivos del diccionario de datos.
- **Frozen**: Evita que las definiciones de campos e índices sean modificadas. No incluyen los registros de datos.
- La opción de **String Attrs**, se utiliza para definir los atributos de las etiquetas de un archivo que va a ser traducido por el Translation Manager.
- **Validaciones**: Establece una expresión que si al ser evaluada, resulta falsa, evita que el archivo sea eliminado.
- **Triggers**: Es un código 4GL que se ejecuta como resultado de un evento de base de datos, como:
borrar y crear archivos.

Ejemplo: Trigger Procedure for Create of articulo

*******Dentro de los Triggers se manejan las siguientes opciones:**

- **Check CRC**: Proceso de validación que verifica la integridad de la base de datos para poder ejecutar un trigger.
- **Overdrable**: Permite que un trigger sea o no modificado.

Campos

Los campos son los atributos de las tablas. En el diccionario de datos es posible definirles, entre otras, las siguientes características:

Crear campos:

The screenshot shows a Windows-style dialog box titled "Create Field for Table articulo". It contains several input fields and checkboxes. The "Field Name" field is empty. The "Data Type" dropdown is set to "CHARACTER". The "Format" field is set to "X(8)". The "Label" field is empty. The "Column Label" field is empty. The "Initial Value" field is empty. The "Order #:" field is set to "1.0". The "Decimals:" field is set to "1". The "Position:" field is empty. The "Description:" field is empty. The "Help Text:" field is empty. There are three checkboxes: "Mandatory" (unchecked), "Case-Sensitive" (unchecked), and "Extent" (checked, with a value of "0" next to it). At the bottom of the dialog are several buttons: "Triggers...", "Validation...", "View As...", "String Attrs...", "Data Source...", "OK", "Create", "Cancel", and "Help".

- ✓ Nombre
- ✓ Tipo de dato
- ✓ Formato de captura
- ✓ Etiqueta
- ✓ Etiqueta para desplegado
- ✓ Valor inicial
- ✓ Orden del campo
- ✓ Descripción
- ✓ Ayuda
- ✓ Mandatory
- ✓ Case Sensitive
- ✓ Extent

Propiedades de los campos:

- Carácter: Contiene datos alfanuméricos.
- Integer: Acepta números enteros positivos o negativos. Rango:2,147,483,648 a 2,147,483,647.
- Decimal: Es un dato compuesto de dígitos, incluyendo 10 dígitos a la derecha del punto decimal.
- Logical: Imacena valores YES/NO ó TRUE/FALSE.
- Date: Representa datos de fechas.
- Recid : macena valores de identificadores de registro.
- Mandatory: Indica que el campo debe de ser capturado obligatoriamente
- Case Sensitive: Es utilizado solamente para campos de tipo carácter. Si es falso permite realizar comparaciones entre caracteres sin importar la forma en que son escritas.
Ejemplo: Clave = clave = CLAVE
- Extent: Indica que el campo es un arreglo de múltiples elementos.

Para realizar Índices:

¿Qué es un Índice?

Es un dato elemental de un registro que se utiliza como criterio de identificación.

Razones para definir un índice:

- ✓ Identificación única
- ✓ Ordenamiento de registros
- ✓ Relación con otras tablas
- ✓ Rápido acceso a registros

Tipos de Índices

- **Primario:**

Establece por default que el archivo será ordenado por este campo a menos que se especifique el ordenamiento por otro índice.

- **Activo:**

Indica si un índice esta activo o no. En el caso de que se establezca un índice en un archivo muy largo es más eficiente definirlo como inactivo y solamente activarlo cuando sea necesario.

- **Único:**

Indica que el contenido del índice debe de ser diferente.

Ej. R.F.C., Matricula, Curp, etc.

- **Word index:**

Permite acceder búsquedas por palabras o por frases sobre el campo.

- **Abbreviated:**

Permite que el registro sea accedido utilizando los primeros caracteres del Campo. Se utiliza solamente en campos de tipo carácter.

- Add

Sirve para declarar como índice el campo que se encuentra seleccionado en el rectángulo del lado izquierdo. Al hacer clic sobre este botón, dicho campo pasa al rectángulo del lado derecho.

- Remove

Sirve para eliminar el índice seleccionado en el rectángulo derecho. Al hacer clic sobre este botón, se elimina dicho índice y pasa al campo al rectángulo del izquierdo.

- Move Down:

Sirve para mover el índice seleccionado una posición abajo.

- Move Up:

Sirve para mover el índice seleccionado una posición arriba.

Reglas para la validación de datos

Se usa para definir una expresión y mensajes de validación para la tabla.

1. Se selecciona la tabla o campo para el cual se quiere establecer el criterio de validación.
2. Seleccione la opción Validation .
3. Una vez desplegada la caja de dialogo, escribir la expresión que debe ser verdadera para que:
 - Se puede borrar un registro de la tabla.
 - Que valor deberá ser introducido en ese campo.
4. Escribir el mensaje que le diga a los usuarios que hacer cuando intentan borrar o eliminar una entrada.
5. Seleccionar OK.

Secuencias

Una secuencia es un objeto de base de datos que proporciona valores incrementales a una aplicación. Puede generar valores secuenciales dentro de cualquier rango de enteros, con valores positivos o negativos.

Como crear una secuencia

1. Desde la ventana de data dictionary, se selecciona el icono de Sequense para que se despliegue la ventana de Create Séquense (nótese que las etiquetas de los botones han cambiado), en donde se encuentran los botones Create Sequense, Séquense Propieties y Delete Séquense.
2. Cuando se haya desplegado la ventana anterior, se hace clic en el botón de Create Séquense, es el único que se encuentra activado cuando se despliega por primera vez. El botón central, se usa para ver las propiedades de una secuencia que anteriormente haya sido creada; estas propiedades se pueden cambiar. El botón de la derecha, se utiliza para eliminar la secuencia que se encuentra seleccionada.

3. Dentro de la caja de dialogo, se escribe el nombre de la secuencia, el valor inicial, el incremento asi como el superior. Adicionalmente se puede indicar que haga un ciclo(que vuelva a iniciar) al llegar al limite.
4. Cuando se haya terminado de crear la secuencia, se puede hacer clic en el botón de Create para crear otra secuencia, o hacer clic en el botón de Cancel para eliminar la secuencia actual. Para salir de esta caja de dialogo y aceptar las secuencias, se hace clic en el botón de Ok

¿Qué es el Administrador de Base de Datos?

Es una herramienta creada por PROGRESS que auxilia al usuario en la administración de la base de datos.

Principales funciones:

- ✓ Dump y Load de datos.
- ✓ Exportar e Importar datos.
- ✓ Seguridad en base de datos.

Proceso Dump

Genera archivos que pueden contener definiciones, datos de los archivos de la base de datos.

Principales opciones:

- Data definition : genera un archivo(.df), contiene el esquema.
- Table contents: genera un archivo (.d), contiene datos.

Proceso Load

Reconstruye una base de datos archivos Dump previamente generados.

Principales opciones:

- Data definition: Reconstruye un archivo basado en el esquema almacenado en el archivo (.df).
- Table contents: Genera los datos de un archivo basado en los datos almacenados en el archivo (.d).

EL EDITOR DE PROCEDIMIENTOS

El editor de procedimientos (Procedure Editor) es un editor de textos en donde se escriben los procedimientos de PROGRESS. Se puede invocar directamente desde el escritorio de ADE. Cuando se inicia abre un buffer sin nombre. Cada buffer en el editor de procedimientos puede contener: un procedimiento recién creado, o uno que anteriormente haya sido creado.

PROGRESS permite mantener abiertos varios buffers a la vez, lo que es muy conveniente ya que cuando se trabaja con varios buffers a la vez, lo que es muy conveniente ya que cuando se trabaja con varios de ellos, se puede transferir información entre los mismos mediante las opciones de copiar, cortar y pegar.

Cómo editar texto

La navegación en el Editor de Procedimientos bajo Windows es similar a la navegación en cualquier otro editor de texto. Se puede usar tanto el mouse como el teclado para desplazar el punto de inserción, o para marcar el texto, o para ir directamente a una línea de texto determinada. El editor de procedimientos también permite compilar, depurar y correr los procedimientos. En la siguiente tabla se muestran las teclas rápidas para algunas opciones del menú:

TECLA	F1	F2	F3	F4	F6	F7	F8	F9
FUNCION	Help	Run	Open		Save	Buffer Siguiente	Close	Encontrar siguiente
SHIFT + TECLA		Revisar syntaxis	New	Debug	Guardar Como	Buffer Anterior		Encontrar anterior
CTRL + TECLA			Ventana de procedimiento nueva					

Teclas de función disponibles en el editor de procedimientos de PROGRESS

TECLA	X	C	V	F	R	G	L	E
CTRL.+	Cut	Copy	Paste	Find	Replace	Go to line	List Buffers	Mensajes de compilador

Combinación de teclas para la edición de un procedimiento desde el Procedure Editor

Para crear un archivo nuevo p se debe comenzar en la opción new de menú file o usar la combinación de teclas <shift><F3>.

Cuando se ha terminado de escribir el código de nuevo procedimiento en el buffer, se debe guardar para crear un nuevo archivo con la extensión.p. Para hacer esto, se usa la opción save del menú file o <F6>. Se debe agregar la extensión .p en el nombre del archivo ya que al abrirlo en otra ocasión posterior, PROGRESS emplea un filtro en donde se muestran todos los archivos .p.

Para abrir un archivo .p creado anteriormente, se puede usar la opción open del menú file o la tecla <F3>.

- Cuál procedimiento se va a compilar
- Si se quiere guardar el archivo compilado como un archivo permanente .r o en un archivo de sesión compilada (SCV)
- Las diferentes listas

Para iniciar la compilación de un procedimiento con el estatuto COMPILE, se corre el procedimiento que contiene a dicho estatuto. También se puede compilar cuando se encuentra desplegado usando la opción Application Compiler del menú Tools. Esta opción permite iterativamente definir los parámetros.

El empleo de la opción Compiler del menú Options permite definir algunos parámetros opcionales del comando compile de la misma manera que se haría con es estatuto COMPILE.

CAPÍTULO V

SINTAXIS DEL LENGUAJE DE PROGRESS

PROGRESS es un lenguaje de cuarta generación.

PROGRESS posee una sintaxis muy parecida al inglés hablado y proporciona estatutos muy poderosos para:

- Tener acceso ala Base de Datos
- Programar interfaces de usuarios (UI)

Así mismo, proporciona todo el flujo y control para la construcción típica de un lenguaje de tercera generación (3GL).

Elementos del lenguaje

En la siguiente tabla se listan los componentes de la sintaxis del lenguaje de PROGRESS.

Elementos del lenguaje	Descripción
Statements	Definen una acción o proceso
Variables	Guardan datos en memoria para su procesamiento
Elementos de base de datos	Guardan datos en la base de datos
Operadores	Símbolos que ejecutan operaciones matemáticas o comparaciones
Expresiones	Combinación de variables, campos de base de datos y operadores, que juntos producen algún valor.
Símbolos especiales	Comentarios, o referencias especiales, referencias tales como archivos include.
Procedimientos	Unidad de ejecución compuesta de una o más instrucciones.
Blocks	Grupo de declaraciones que son parte de la misma rama de condición o de un ciclo
Funciones	Están predefinidas, están dentro de un proceso, tales como cálculos o conversiones
Obetos de interface de usuario	Unidades de despliegue en pantalla, que pueden aceptar datos del usuario, de las variables de la aplicación, de archivos o de la base de datos.

Símbolos especiales

En PROGRESS se utilizan los siguientes símbolos en sus procedimientos:

Símbolo	Papel
/* */	Encierran un comentario
*	Comodín, sustituye a cualquier grupo de caracteres
Punto (.)	• Termina una instrucción • Comodín para un carácter • Separador. Separa el nombre de la base de datos del nombre de la tabla y del nombre del campo
Dos Puntos (:)	Terminación de Block, encabezado o etiqueta de Block. Introduce una atributo a la sintaxis de traducción

	Introduce un atributo a un objeto o método
?	Designa un valor desconocido
~	Causa que el siguiente carácter sea tratado como literal
“ ” o ‘ ’	Encierra a una cadena de caracteres
{nombre-archivo}	Incluye referencia de archivo
{ }	Argumentos de paso
[]	Referencia a una matriz, se usa para subindicar la matriz y los valores iniciales
{&}	Variable del preprocesador

Objetos de interfase de usuario

Un objeto de interfase de usuario (UI) permite desplegar gráficamente los datos en la pantalla. La siguiente tabla lista algunos de los elementos que se pueden usar cuando se trabaja con objetos:

Elemento	Descripción
Object	Elemento básico o componente de la GUI
Events	Acción emprendida por el usuario sobre algún objeto
Trigger	Respuesta programada para un evento.
Handle	Indicador nombrado para un objeto.
Attribute	Alguna característica de un objeto ya sea que se puede leer o asignar diferente valor
Method	Acción sobre algún objeto que regresa algún valor

Componentes de los estatutos

La sintaxis de los estatutos de PROGRESS esta formada por una combinación de: Una palabra clave (keyword) obligatoria, parámetros y opciones específicas del estatuto, frases, expresiones y un terminador de estatuto que generalmente es un punto (.)

Componente	Papel
Keyword	Es el verbo principal de la declaración, designa la acción o el proceso que se ha de ejecutar
Parametre(s) and Option(s)	Información adicional que indica que algo esta cambiando la acción o proceso designado por la declaración.
Phrase(s)	Algún elemento común del lenguaje significa que pueden aparecer en algunas declaraciones de PROGRESS
Expression	Es una combinación de variables, constantes y operadores que evalúan a un elemento de

El comando DISPLAY mueve los datos del buffer de registros al buffer de la pantalla (se despliegan los valores de las variables en la pantalla)

Sintaxis

DISPLAY

Expresión [format-phrase]
SPACE [(N)]
SKIP [(n)]

[frame-phrase]

En la siguiente tabla se indica el significado de cada uno de los parámetros que se emplean con el estatuto DISPLAY:

Parámetro	Designa
Expresión	Lo que se despliega. Puede ser una variable, una constante o un campo de base de datos
Format-phrase	Como se despliega el dato
Frame-phrase	Características del marco que contiene los elementos desplegados.
SPACE [(N)]	Número de espacios que se despliegan. Puede ser cero o un número positivo. Si es mayor el espacio del que cabe en la línea, PROGRESS salta a la siguiente línea y no despliega más espacios.
SKIP [(N)]	Número de líneas que debe saltar: • Si n es cero o si no se especifica , PROGRESS salta a la siguiente línea, a menos que ya haya saltado a una nueva línea debido a la longitud de los datos desplegados. • Si se especifica SKIP (1) PROGRESS salta a una nueva línea sin importar que ya haya saltado una nueva línea.

Introducción de valores para las variables

Estatutos para aceptar entradas del usuario

Para permitir al usuario introducir datos se usan los siguientes estatutos SET, PROMPT-FOR Y UPDATE.

PROMPT-FOR

Interactúa solamente con el buffer de pantalla, permite al usuario introducir valores pero no los pasa al buffer de registro por lo que no se pueden tomar valores para usarlos ya que dichos valores permanecen en el buffer de pantalla solamente. Para poder usar estos valores se debe usar la función INPUT.

Ejemplo:

```
/* p5-13 */
DEFINE VARIABLE nombre AS CARÁCTER FORMAT "X (20)"
    LABEL "Introduzca su nombre".
DEFINE VARIABLE saludo AS CARÁCTER FORMAT "X (27)"
PROMPT-FOR nombre.
ASSIGN saludo = "Hola, " + INPUT nombre + "!" .
DISPLAY saludo.
```

El estatuto SET

Se le puede pedir al usuario que introduzca un valor , que será asignado a una variable (PROGRESS lo pasa del buffer de pantalla al buffer de registro)

Ejemplo:

```
/* p5-12 */
DEFINE VARIABLE name AS CARÁCTER FORMAT "X (20)"
    LABEL "Introduzca su nombre".
DEFINE VARIABLE saludos AS CARÁCTER FORMAT "X (27)"
SET name.
ASSIGN saludos = "Hola, " + name + "!" .
DISPLAY saludos.
```

El estatuto UPDATE

El estatuto UPDATE despliega los campos o variables, solicita al usuario una entrada y posteriormente la coloca tanto en el buffer de pantalla como en los campos o variables específicos. Es una combinación de los siguientes estatutos:

- DISPLAY ya que pasa los valores de los campos o variables al buffer de pantalla y los despliega.
- PROMPT-FOR ya que solicita al usuario datos y los coloca en el buffer de pantalla.
- ASSIGN porque pasa los datos del buffer de pantalla al buffer de registro.

Sintaxis

UPDATE

```
[{field [format-phrase][WHEN expression]}|  
 {TEXT (field [format-phrase]...)}|  
 {field = expression }| {constant [ A T n|TO n]}|  
 {^}|{SPACE [(n)]}|{SKIP[(n)]}|....  
[GO-ON (key-label...)] [frame-phrase] [editing-phrase]  
[NO-ERROR]
```

Ejemplo de sintaxis

```
UPDATE record [EXCEPT field...] [frame-phrase] [NO-ERROR]
```

En el siguiente ejemplo, el procedimiento permite desplegar cada registro de la tabla customer así como modificar el nombre y el límite de crédito. LA opción VALIDATE en le primer estatuto UPDATE garantiza que se introduzcan valores menores de 500,000. La opción HELP despliega un mensaje para tal efecto.

```
/* r-updat2.p */
```

```
FOR EACH customer:
```

```
    UPDATE customer.name
```

```
        Credit-limit VALIDATE (credit-limit < 500000, "Too high")
```

```
        HELP "Enter credit-limit < 500000".
```

```
    FOR EACH order OF customer:
```

```
        DISPLAY order-num.
```

```
        UPDATE promise-date ship-date VALIDATE(ship-date <
```

```
today,
```

```
        "Ship date must be later than today ").
```

```
    END
```

```
END
```

El segundo block lee cada registro de la tabla order que pertenece a customer, despliega el campo order-num y permite modificar los campos promise-date y ship-date. La opción VALIDATE asegura que no se introduzca un fecha de envío anterior a la actual. **Este procedimiento.....sigue**

Las Expresiones PROGRESS

Las expresiones son parecidas a una formula en donde se combinan variables y/o constantes junto con algunos operadores. Los elementos que se pueden usa son: constantes, variables, campos de base de datos atributos de los objetos y operadores.

	expresión	
--	-----------	--

En la siguiente tabla se describen algunas de las funciones y operaciones que se pueden aplicar a un *string*: contiene una lista de elementos. La lista se delimita por comas por omisión, sin embargo, se puede indicar otro delimitador.

Función	Sintaxis	Operación
LOOKUP	LOOCKUP (string, list[,delimitador])	La posición del string en lista.
ENTRY	ENTRY (index, list[delimitador])	El carácter es índice en el string.
NUM-ENTRIES	NUM-ENTRIES (list([,delimitador]))	Número de elementos en la lista.

Funciones para fechas

Las funciones relacionadas con las fechas permiten obtener la fecha actual o manipular la información de fechas. En la siguiente tabla se hace una descripción de las funciones de fecha:

Función	Sintaxis	Operación
Today	TODAY	Regresa la fecha actual del S. O.
Date	DATE(mes, día, año /fecha-string)	Convierte un string a tipo date.
Day	DAY (fecha-expresión)	Regresa el número correspondiente al día.
Month	MONTH(fecha-expresión)	Regresa el número correspondiente al mes.
Weekday	WEEKDAY(fecha-expresión)	Regresa el string correspondiente al día.
Year ()	YEAR(fecha-expresión)	Regresa el número correspondiente al año.
Time ()	TIME	Regresa la hora del sistema operativo.

Ejemplo:

/* Operaciones con la función DATE*/

```
DEFINE VARIABLE days_since AS INTEGER FORMAT "9999".
day_since = TODAY – DATE ("01 / 01 / 2005 ").
DISPLAY "Hoy es " TODAY.
DISPLAY "Hoy es el " days_since "avo. Día del año." WITH NO-LABELS.
```

Funciones numéricas y operadores

Junto con las operaciones aritméticas se pueden usar la operación TRUNCATE y el operador MODULE para operar con valores numéricos, también se pueden convertir otros tipos de datos a tipos de datos numéricos. En la siguiente tabla se indica el uso de las funciones:

Función	Sintaxis	operación
INTEGER	INTEGER(expresión)	Convierte a la expresión en enteros tipo fecha.
DECIMAL	DECIMAL(expresión)	Convierte la expresión a dato tipo decimal.
TRUNCATE	TRUNCATE(decimal, num-dec)	Trunca num-dec de decimal
MODULO	Num1 MODULE num2	Divide num1 entre num2 y regresa el residuo.

Usando Set

Se puede utilizar el SET para vaciar pronto algunos valores, y entonces estos valores son asignados a las variables.(Progress mueve el registro del buffer). El siguiente ejemplo permite la entrada hasta el primer nombre.

```
/* Programa que ejemplifica el uso del SET */

DEFINE VARIABLE nombre AS CHARACTER FORMAT "X(20)"
LABEL "Escribe tu primer nombre".
DEFINE VARIABLE saludo AS CHARACTER FORMAT "X(27)".
SET nombre.
ASSIGN saludo = "Hola, " + nombre + " ! ".
DISPLAY saludo.
```

Usando el Prompt-For

PROMPT-FOR permite el uso de la entrada de valores, pero no permite mover el valor entre el buffer de registro. Por eso tú no puedes referirlo, desde que un valor entra en el buffer de pantalla tu lo puedes referir al uso de la función INPUT.

```
/* Programa que ejemplifica el uso del PROMPT-FOR */

DEFINE VARIABLE nombre AS CHARACTER FORMAT "X(20)"
LABEL "Escribe tu primer nombre".
DEFINE VARIABLE saludo AS CHARACTER FORMAT "X(27)".
PROMPT-FOR nombre.
INPUT nombre.
ASSIGN saludo = "Hola, " + nombre + " ! ".
DISPLAY saludo.
```

NOTA: Poner INPUT.

ASSIGN saludo = “Hola, ” + INPUT nombre + “! “.

BLOQUES EN PROGRESS

¿ Qué es un bloque ?.

Una serie de estatutos que PROGRESS trata como una sola unidad para definirles características.

-Generalmente un bloque comienza con un estatuto de encabezado y concluye con un estatuto END.

Tipos de Bloques

Procedimientos:

- Procedimiento principal.
- Procedimiento interno.
- Procedimiento externo (programas).

Triggers.

Bloques de control:

- DO.
- REPEAT.
- FOR EACH.

Reglas generales para bloques.

- Siempre inician con un estatuto de encabezado de bloque.
- Siempre terminan con un estatuto END.
- Tienen servicios implícitamente definidos llamados propiedades de bloque.
- Algunos bloques pueden ser nombrados.
- A varios bloques es posible definirles servicios de forma explícita.

Bloques de Control

-

- Bloque de DO.

Es usado generalmente para agrupar estatutos y éstos sean tratados como una sola unidad.

Sintaxis parcial:

{ label : }

DO { FOR record }

{ variable = expression 1 TO expression 2 { BY k } }

{ WHILE expression }:

```

Def var char1 as format "x(5)" case-sensitive initial "hola".
Def var char-arr1 as char format "q9" extent 4 initial [1, 2, 3, 4].
Def var log1 as logical format "In/out" initial "in".

```

```

Assign dia1 = TODAY.

```

```

DISPLAY "aquí están los valores de hoy:"      TO 40
    dec1 format "99.99"                       TO 40
    int1 format "9"                           TO 40
    dia1                                       TO 40
    char1                                      TO 40
    char-arr1[1]                              TO 40
    char-arr[2]                               TO 40
    char-arr[3]                               TO 40
    log1                                       TO 40

```

```

WITH SIDE-LABELS CENTERED.

```

Colocar usar las opciones AT y COLON

Con la opción AT de la format phrase, se puede indicar un lugar exacto para cada elemento dentro del marco. También se puede indicar el alineamiento, ya sea al a izquierda, derecha o los dos puntos para separar los datos de su etiqueta en la posición indicada con la opción AT. Si se desea, se pueden alinear los separadores dos puntos de los elementos dentro del marco usando la opción COLON en la format phrase.

La palabra clave AT se refiere al marco y no a la ventana. Para cambiar la ubicación del marco dentro de la ventana se usan las palabras clave CENTERED o ROW y COLUMN (después de la palabra clave WITH).

/*Ejemplo*/

```

def var dec1 as decimals 1 initial 3.1.
def var int1 as int initial 0.
Def var int-arr1 as int 3 initial [1, 2, 3] format "9".
Def var deal as date.
Def var char1 as char format "x(5)" case-sensitive initial "Hola".
Def var char-arr1 as char format "Q9" Extent 4 initial [1, 2, 3, 4].
Def var log1 as logical format " In / out" initial "in".

```

```

Assign dia1 = today.

```

```

Display "aquí están los valores de hoy:" at column 30 row 1 right-aligned
    dec1 format "99.99" at column 30 row 2 right-aligned
    int1 format "9" at column 30 row 3 right-aligned
    int-arr[1] at column 30 row 4 right-aligned
    int arr[2] at column 30 row 5 right-aligned
    int-arr[3] at column 30 row 6 right-aligned
    dia1 at column 30 row 7 left-aligned
    char1 at column 30 row 8 left-aligned

```

```
char-arr[1] at column 30 row 9 colon-aligned
char-arr[2] at column 30 row 10 colon-aligned
char-arr[3] colon 30
log1 colon 30
with side-labels centered.
```

Como definir un marco

Con el uso de la frame-phrase junto con el uso de estatuto DISPLAY u otros estatutos se pueden definir objetos- marco que posteriormente pueden ser usados en estatutos que despliegan datos. El estatuto DEFINE-FRAME le da un nombre a un marco y define sus características de despliegue. Posteriormente se pueden desplegar los elementos dentro del marco al hacer referencia a él a través del nombre indicado en éste estatuto.

Sintaxis

```
DEFINE-FRAME nom-marco [form-item][frame-phrase]
```

Form-item

```
field[format-phrase]
variable [at-phrase]
SPACE[n]
SKIP[n]
```

Como hacer referencia a un marco con nombre

Una vez que se haya definido un marco, se puede usar el siguiente formato para frame-phrase junto con el estatuto DISPLAY:

```
DISPLAY ..... UIT[FRAME nom-marco]
```

Como usar un marco definido

Existen varias razones para usar marcos con nombre. Se tiene control explícito sobre las cuales los mismos elementos de marco se muestran; una vez creada la plantilla del marco, se puede usar par desplegar los mismo elementos en el mismo formato en diferentes lugares del código, sin tener que repetir la definición del despliegue; se puede reusar la definición del marco en más de un procedimiento; se puede definir y usar más de un marco en un procedimiento, mostrando los mismos campos diferentes veces en diferentes posiciones en la misma pantalla.

Si se usa el marco por omisión que proporciona PROGRESS para el bloque de procedimiento, no se puede lograr la salida deseada. Si se indica el mismo nombre de la variable más de una vez en el mismo marco, PROGRESS despliega todas las ocurrencias de la variable en la misma posición.

/*Ejemplo*/

```

Def var nombre as char format "x(10)" label "Nombre" initial "Juan".
Def var apellido as char format "x(15)" label "Apellido" initial "Pérez".
Def var cia as char format "x(15)" label "compañía " initial "PROGRESS".
Display nombre at row 1 column 8
      apellido at row 1 column 15
      cia row 1 column 25
      with centered.
Display
      nombre at row 5 column 8
      "trabaja en " at row 5 column 15
      cia          at row 5 column 25
                  with no-labels centered no-box row 2.

```

Para obtener las salida deseada, se utilizan dos marcos, uno para desplegar las variables en una columna y otra para mostrarlas en forma de oración. Se usan dos DISPLAY para referirse a esos marcos, indicando cual variable va dentro de cual marco para un despliegue determinado.

/*Ejemplo/

```

Def var nombre as char format "x(10)" label "Nombre" initial "Juan".
Def var apellido as char format "x(15)" label "Apellido" initial "Pérez".
Def var cia as char format "x(15)" label "compañía " initial "PROGRESS".

Define frame f-nombre
      nombre colon 15
      apellido colon 15
      cia colon 15
      with side-labels row 1 centered.
Define frame f-cia
      nombre at row 2 column 8
      " trabaja en " at row 2 column 15
      cia at row 2 column 25
      with no-labels centered no-box.
Display nombre apellido cia with frame f-nombre.
Display nombre cia with frame f-cia.

```